UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/713,633	11/14/2003	Bryan M. Cantrill	03226.342001; SUN040194	7049

32615 7590 07/12/2007

OSHA LIANG L.L.P./SUN
1221 MCKINNEY, SUITE 2800
HOUSTON, TX 77010

EXAMINER	
NGUYEN, PHILLIP H	

ART UNIT	PAPER NUMBER
2191	

MAIL DATE	DELIVERY MODE
07/12/2007	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	10/713,633	CANTRILL, BRYAN M.	
	Examiner	Art Unit	
	Phillip H. Nguyen	2191	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *16 April 2007*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-36* is/are pending in the application.

4a) Of the above claim(s) *3 and 20* is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1,2,4-19 and 21-36* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

1. This action is in response to the amendment filed on 4/16/2007. Claims 3 and 20 have been cancelled. Claims 1, 2, 4-19 and 21-36 remain pending and have been considered below.

Claim Objections

2. The amendment filed on 4/16/2007 overcomes the objection set forth to claims 1, 5, 9, 13, 17, 19, 23, 27, 31 and 35 of previous action. Therefore, the objection is withdrawn.

Claim Rejections - 35 USC § 112

3. The amendment filed on 4/15/2007 overcomes the rejection set forth to claims 1-36 of previous action. Therefore, the rejection is withdrawn.

Claim Rejections - 35 USC § 101

4. The amendment filed on 4/16/2007 overcomes the rejection set forth to claims 1-18 or previous action. Therefore, the rejection is withdrawn. Applicant did not amend claim 19 to overcome the software per se rejection. Therefore examiner maintains the rejection set forth to claims 19-36.

5. Claim 19 is non-statutory because it recites a system but appears reasonable to interpret this system by one or ordinary skill in the art as software per se. Applicant's specification provides no explicit and deliberate definition of the components

("instrumented program", "a thread", "a trap handler") that make up the system other than they could be software components, which are directed to functional descriptive material, per se, and are therefore non statutory. Applicant is suggested to amend this claims to include at least a processor or a storage device to overcome the rejection. Claims 21-36 directly or indirectly depend on claim 19 and therefore have been addressed in connection with the rejection set forth to claim 19.

Response to Arguments

6. Applicant's arguments filed 4/16/2007 have been fully considered but they are not deemed persuasive.

Applicant asserts on page 14 of the amendment that Yates fails to disclose instrumented program that includes at least one trap instruction.

Examiner respectfully disagrees with all the allegations as argued. Instrumentation is a process of adding code to the program as well as probing process. Probing is a processing of adding code or replacing original code with probe instruction for monitoring the program flow. A probe instruction is considered as a trap instruction. It seizes controls away from the X86 binary (see at least col. 84, lines 54-67). Therefore, an instrumented program is also considered as a probed program.

Applicant asserts on page 15 of the amendment that Yates fails to disclose functionality to halt execution of a thread when a trap instruction is encountered during tracing of an instrumented program. Applicant further asserts on page 15 that Yates also fails to disclose tracing.

Examiner respectfully disagrees with the allegation as argued. When a probe instruction seizes controls away from the X86 binary, the execution of X86 binary (instructions) is halted (see at least col. 84, lines 54-67). Yates further discloses profiling of an X86 program (see at least col. 84, lines 55-65). According to Yates, profiling is a processing of recording information regarding to the execution of a program in order to analyze the performance and behavior of the program (also see at least col. 7, lines 20-25, 50-53, 60-65). Tracing is a processing of gathering information of the program. Therefore, tracing and profiling are similar process.

Applicant asserts on page 16 of the amendment that Examiner did not address the limitation of claim 2.

Examiner respectfully disagrees with the allegation. Examiner on his previous action did address the limitation of claim 2.

Examiner is entitled to give claim limitations their broadest reasonable interpretation consistent with the specification. Applicant always has the opportunity to amend the claims during prosecution and broad interpretation by the examiner reduce the possibly that the claim, once issued, will be interpreted more broadly than is justified. In re Prater, 162 USPQ 541, 550-51 (CCPA 1969).

Claim Rejections - 35 USC § 102

1. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

A person shall be entitled to a patent unless –

(e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

2. Claims 1-36 are rejected under 35 U.S.C. 102(e) as being anticipated by Yates, Jr. et al. (United States Patent No.: US 7,065,633 B1).

As per claim 1:

Yates discloses:

- triggering a trap instruction in the instrumented program during tracing of the instrumented program (see at least col. 23, lines 10-11 "**an exception occurring during execution of a program coded in the CISC instruction set**"; also see at least col. 23, lines 25-27; "**the exception may be a trap requesting a file access service form the CISC operating system on behalf of the program**");
- transferring control of the instrumented program to a trap handler associated with the trap instruction (see at least col. 51, lines 39-40 "**the interrupt is delivered to emulator 316, which handles the interrupt**"); and
- calling into a tracing framework by the trap handler to perform a tracing operation associated with the trap instruction (see at least col. 32, lines 7-9 "**during emulation of the X86 program, prober 600 monitors the program flow for execution of X86 instructions that have been translated into native code**");
- performing the tracing operation to obtain tracing information, wherein the tracing information is used to analyze the instrumented program (see at least col. 84,

lines 56-58 "**Profiler 400 generates a profile of an X86 program. Hot spot detector 122 analyzes the profile to identify often executed sections of code**");

- emulating, after performing the tracing operation, an original instruction in the instrumented program using the trap handler instruction (see at least col. 23, lines 13-15 "**the RISC operating system may include a collection interrupt service routines programmed to emulate instructions in the CISC instruction set**"; also see at least col. 51, lines 34-42 "**when a high-priority X86 interrupt interrupts X86 emulator 316 while emulating a complex instruction...the interrupt is delivered to emulator 316, which handles the interrupt**"), wherein the original instruction is associated with the trap instruction and wherein the original instruction relates to creating or dismantling a stack frame instruction (see at least col. 40, lines 16-17 "**if the program creates code in writable storage (stack or heap) on the fly**").

As per claim 2:

Yates further discloses:

- replacing the instruction with the trap instruction in the instrumented program (see at least col. 30, line 24 "**Probing**" – **Probing is a process of replacing/adding instruction(s) with probe instruction(s) for monitoring the program flow**).

As per claim 4:

Yates discloses:

- wherein emulating the instruction comprises emulating a push1 instruction (see at least col. 132, lines 23-25 "**any asynchronous interrupts that occur during the PUSHA converter recipe are handled in tapestry operating system 312 or in emulator 316**").

As per claim 5:

Yates discloses:

- wherein emulating a push1 instruction comprises:
    - obtaining a stack pointer location, wherein the stack pointer location corresponds to a location in the stack frame (see at least col. 132, lines 7-9 "**the eight STOREDEC instructions 951 push the eight general purpose registers EAX, ECX, EDX, EBX, ESP, EBP, ESI and ECI onto the stack**");
    - incrementing an instruction pointer to obtain an incremented instruction pointer (see at least col. 122, lines 38-40 "**the instruction pointer (IP) value is incremented by the length count after each instruction that is marked with an end-of-recipe marker**");
    - loading the incremented instruction pointer in the stack frame at one location before the stack pointer location (see at least col. 136, lines 7-10 "**STOREDEC, is a pre-decrementing store that pushes the IP value**

onto the stack segment (SS) at the offset specified by the stack pointer ESP");

- loading a code segment (CS) value stored one location after the stack pointer location into the stack pointer location (see at least col. 135, lines 48-49 "**the current offset into the code segment (EIP) is pushed onto the stack**");
- loading an EFLAGS values stored two locations after the stack pointer location into one location after the stack pointer (see at least col. 91, lines 13-16 "**whether addressing is physical or virtual, the floating point stack pointer, and the full/empty state of floating-point register, and operand sizes are encoded in segment descriptors, the EFLAGS register**"); and
- loading a base pointer into two location after the stack pointer location (see at least col. 132, lines7-9 "**the eight STOREDEC instructions 951 push the eight general purpose registers EAX, ECX, EDX, EBX, ESP, EBP, ESI and ECI onto the stack**").

As per claim 6:

Yates discloses:

- decrementing the stack pointer location by one location (see at least col. 132, lines 43-44 "**the register carries the intermediate decrementing of the stack pointer**"); and

- issuing a return from interrupt instruction (see at least col. 94, lines15-16 "**a trap instruction transfers control to an exception handler**").

As per claim 7:

Yates discloses:

- wherein the instruction pointer is loaded at the stack pointer location (see at least col. 136, lines 7-10 **STOREDEC, is a pre-decrementing store that pushes the IP value onto the stack segment (SS) at the offset specified by the stack pointer ESP**").

As per claim 8:

Yates discloses:

- emulating the original instruction comprises emulating a enter instruction (see at least col. 132, lines 41 "**PUSHA**"; also see at least col. 132, line 36; **MOV instruction**").

As per claim 9:

Yates discloses:

- obtaining a stack pointer location, wherein the stack pointer location is corresponding to a location in the stack frame (see at least col. 132, lines 7-9 "**the eight STOREDEC instructions 951 push the eight general purpose registers EAX, ECX, EDX, EBX, ESP, EBP, ESI and ECI onto the stack**");

- incrementing an instruction pointer to obtain an incremented instruction pointer (see at least col. 122, lines 38-40 "**the instruction pointer (IP) value is incremented by the length count after each instruction that is marked with an end-of-recipe marker**");
- loading the incremented instruction pointer in the stack frame at one location before the stack pointer location (see at least col. 136, lines 7-10 "**STOREDEC, is a pre-decrementing store that pushes the IP value onto the stack segment (SS) at the offset specified by the stack pointer ESP**");
- loading a code segment (CS) value stored one location after the stack pointer location into the stack pointer location (see at least col. 135, lines 48-49 "**the current offset into the code segment (EIP) is pushed onto the stack**");
- loading an EFLAGS values stored two locations after the stack pointer location into one location after the stack pointer (see at least col. 91, lines 13-16 "**whether addressing is physical or virtual, the floating point stack pointer, and the full/empty state of floating-point register, and operand sizes are encoded in segment descriptors, the EFLAGS register**");
- loading a base pointer into two location after the stack pointer location (see at least col. 132, lines 7-9 "**the eight STOREDEC instructions 951 push the eight general purpose registers EAX, ECX, EDX, EBX, ESP, EBP, ESI and ECI onto the stack**"); and

- loading the base pointer into a base pointer register (see at least col. 34, lines 9-10 "**EBP register of the X86 architecture are mapped by converter hardware 136 to register R53**").

As per claim 10:

Yates discloses:

- decrementing the stack pointer location by one location (see at least col. 132, lines 43-44 "**the register carries the intermediate decrementing of the stack pointer**"); and
- issuing a return from interrupt instruction (see at least col. 15-16 "**a trap instruction transfers control to an exception handler**").

As per claim 11:

Yates discloses:

- wherein the instruction pointer is loaded at the stack pointer location (see at least col. 136, lines 7-10 "**STOREDEC, is a pre-decrementing store that pushes the IP value onto the stack segment (SS) at the offset specified by the stack pointer ESP**").

As per claim 12:

Yates discloses:

- wherein emulating the instruction comprises emulating a pop1 instruction (see at least col. 58, line 1 "**when an X86 POPF instruction is emulated**").

As per claim 13:

Yates discloses:

- obtaining a stack pointer location, wherein the stack pointer location corresponds to a location in the stack frame (see at least col. 132, lines 7-9 "**the eight STOREDEC instructions 951 push the eight general purpose registers EAX, ECX, EDX, EBX, ESP, EBP, ESI and ECI onto the stack**");
- loading the base pointer obtained from three location after the stack pointer location into a base pointer register (see at least col. 34, lines 9-10 "**EBP register of the X86 architecture are mapped by converter hardware 136 to register R53**");
- loading an EFLAGS values stored two locations after the stack pointer location into one location after the stack pointer (see at least col. 91, lines 13-16 "**whether addressing is physical or virtual, the floating point stack pointer, and the full/empty state of floating-point register, and operand sizes are encoded in segment descriptors, the EFLAGS register**");

- loading a code segment (CS) value stored one location after the stack pointer location into the stack pointer location (see at least col. 135, lines 48-49 "**the current offset into the code segment (EIP) is pushed onto the stack**");
- incrementing an instruction pointer to obtain an incremented instruction pointer (see at least col. 122, lines 38-40 "**the instruction pointer (IP) value is incremented by the length count after each instruction that is marked with an end-of-recipe marker**"); and
- loading the incremented instruction pointer in the stack frame at one location before the stack pointer location (see at least col. 136, lines 7-10 "**STOREDEC, is a pre-decrementing store that pushes the IP value onto the stack segment (SS) at the offset specified by the stack pointer ESP**").

As per claim 14:

Yates discloses:

- incrementing the stack pointer location by one location (see at least col. 144, lines 65-67 "**when an X86 program changes its stack, both the stack segment descriptor (SS) and the offset may need to be changed**"; also see at least col. 145, line 2 "...**the instruction that loads the stack offset into the stack pointer**"); and
- issuing a return from interrupt instruction ("**a trap instruction transfers control to an exception handler**" Col 94, line 15-16).

As per claim 15:

Yates discloses:

- wherein the instruction pointer is loaded at the stack pointer location (see at least col. 136, lines 7-10 "**STOREDEC, is a pre-decrementing store that pushes the IP value onto the stack segment (SS) at the offset specified by the stack pointer ESP**").

As per claim 16:

Yates discloses:

- wherein emulating the instruction comprises emulating a leave instruction (see at least col. 145, line 18 "**at the end of emulating the move or pop instruction**").

As per claim 17:

Yates discloses:

- obtaining a stack pointer location, wherein the stack pointer location corresponds to a location in the stack frame (see at least col. 132, lines 7-9 "**the eight STOREDEC instructions 951 push the eight general purpose registers EAX, ECX, EDX, EBX, ESP, EBP, ESI and ECI onto the stack**");
- loading the base pointer obtained at the base pointer location into base pointer register (see at least col. 34, lines 9-10 "**EBP register of the X86 architecture are mapped by converter hardware 136 to register R53**");

- loading an EFLAGS values stored two locations after the stack pointer location into one location after the stack pointer (see at least col. 91, lines 13-16 "**whether addressing is physical or virtual, the floating point stack pointer, and the full/empty state of floating-point register, and operand sizes are encoded in segment descriptors, the EFLAGS register**");
- loading a code segment (CS) value stored one location after the stack pointer location into the stack pointer location (see at least col. 135, lines 48-49 "**the current offset into the code segment (EIP) is pushed onto the stack**");
- incrementing an instruction pointer to obtain an incremented instruction pointer (see at least col. 122, lines 38-41 "**the instruction pointer (IP) value is incremented by the length count after each instruction that is marked with an end-of-recipe marker**");
- loading the incremented instruction pointer in the stack frame at one location before the stack pointer location (see at least col. 136, lines 7-10 "**STOREDEC, is a pre-decrementing store that pushes the IP value onto the stack segment (SS) at the offset specified by the stack pointer ESP**"); and
- loading the instruction pointer at three location before the base pointer (see at least col. 135, line 49 "**EIP is pushed onto the stack**").

As per claim 18:

Yates discloses:

- setting the stack pointer location to three locations before the base pointer location (see at least col. 132, lines 7-9 "**the eight STOREDEC instructions 951 push the eight general purpose registers EAX, ECX, EDX, EBX, ESP, EBP, ESI and ECI onto the stack" – the idea is to obtain the stack pointer**);
- incrementing the stack pointer location by one location (see at least col. 44, lines 65-67 "**when an X86 program changes its stack, both the stack segment descriptor (SS) and the offset may need to be changed**"; see at least col. 145, line 2 "**the instruction that loads the stack offset into the stack pointer**"); and
- issuing a return from interrupt instruction (see at least col. 94, lines 15-16 "**a trap instruction transfers control to an exception handler**").

As per claim 19:

Yates discloses:

- the instrumented program comprising at least one trap instruction associated with an original instruction (see at least col. 94, line 15 "**a trap instruction**"), wherein the original instruction relates to creating or dismantling a stack frame (see at least col. 40, lines 16-17 "**if the program creates code in writable storage (stack or heap) on the fly**");
- a thread configured to execute the instrumented program (see at least col. 6, lines 19-20 "**the operating system and the interrupted thread may execute in different instruction set architecture of the computer**");

- a trap handler (see at least col. 125, line 52 "**trap handler**") configured to halt execution of the thread when the trap instruction is encountered during tracing of the instrumented program, call into a tracing framework to perform a tracing operation associated with the trap instruction (see at least col. 32, lines 7-9 "**during emulation of the X86 program, prober 600 monitors the program flow for execution of X86 instructions that have been translated into native code**"), and to emulate the original instruction associated with the trap instruction (see at least col. 23, lines 13-15 **the RISC operating system may include a collection interrupt service routines programmed to emulate instructions in the CISC instruction set**"; also see at least col. 51, lines 34-42 "**when a high-priority X86 interrupt interrupts X86 emulator 316 while emulating a complex instruction...the interrupt is delivered to emulator 316, which handles the interrupt**"); and
- the tracing framework configured to perform the tracing operation to obtain tracing information, wherein the tracing information is used to analyze the instrumented program (see at least col. 84, lines 56-58 "**Profiler 400 generates a profile of an X86 program. Hot spot detector 122 analyzes the profile to identify often executed sections of code**").

As per claim 21:

Yates discloses:

- wherein the instruction is replaced with a trap instruction in the instrumented program (see at least col. 30, line 24 "**Probing" – probing is a process of replacing instruction with probe/trap instruction for monitoring the program flow**).

As per claim 22:

Yates discloses:

- wherein emulating the instruction comprises emulating a push1 instruction (see at least col. 132, lines 23-25 "**any asynchronous interrupts that occur during the PUSHA converter recipe are handled in tapestry operating system 312 or in emulator 316**").

As per claim 23:

Yates discloses:

- obtaining a stack pointer location, wherein the stack pointer location corresponds to a location in the stack frame (see at least col. 132, lines 7-9 "**the eight STOREDEC instructions 951 push the eight general purpose registers EAX, ECX, EDX, EBX, ESP, EBP, ESI and ECI onto the stack**");
- incrementing an instruction pointer to obtain an incremented instruction pointer (see at least col. 122, lines 38-40 "**the instruction pointer (IP) value is**

incremented by the length count after each instruction that is marked with an end-of-recipe marker");

- loading the incremented instruction pointer in the stack frame at one location before the stack pointer location (see at least col. 7-10 "**STOREDEC, is a pre-decrementing store that pushes the IP value onto the stack segment (SS) at the offset specified by the stack pointer ESP**");
- loading a code segment (CS) value stored one location after the stack pointer location into the stack pointer location (see at least col. 48-49 "**the current offset into the code segment (EIP) is pushed onto the stack**");
- loading an EFLAGS values stored two locations after the stack pointer location into one location after the stack pointer (see at least col. 91, lines 13-16 "**whether addressing is physical or virtual, the floating point stack pointer, and the full/empty state of floating-point register, and operand sizes are encoded in segment descriptors, the EFLAGS register**"); and
- loading a base pointer into two location after the stack pointer location (see at least col. 132, lines 7-9 "**the eight STOREDEC instructions 951 push the eight general purpose registers EAX, ECX, EDX, EBX, ESP, EBP, ESI and ECI onto the stack**").

As per claim 24:

Yates discloses:

- decrementing the stack pointer location by one location (see at least col. 132, lines 43-44 "**the register carries the intermediate decrementing of the stack pointer**"); and
- issuing a return from interrupt instruction (see at least col. 15-16 "**a trap instruction transfers control to an exception handler**").

As per claim 25:

Yates discloses:

- wherein the instruction pointer is loaded at the stack pointer location (see at least col. 136, lines 7-10 "**STOREDEC, is a pre-decrementing store that pushes the IP value onto the stack segment (SS) at the offset specified by the stack pointer ESP**").

As per claim 26:

Yates discloses:

- emulating the instruction comprises emulating a enter instruction (see at least col. 132, line 36 "**PUSHA**"; also see at least col. 132, line 41 "**MOV instruction**").

As per claim 27:

Yates discloses:

- obtaining a stack pointer location, wherein the stack pointer location is corresponding to a location in the stack frame (see at least col. 132, lines 7-9 "**the eight STOREDEC instructions 951 push the eight general purpose registers EAX, ECX, EDX, EBX, ESP, EBP, ESI and ECI onto the stack**");
- incrementing an instruction pointer to obtain an incremented instruction pointer (see at least col. 122, lines 38-40 "**the instruction pointer (IP) value is incremented by the length count after each instruction that is marked with an end-of-recipe marker**");
- loading the incremented instruction pointer in the stack frame at one location before the stack pointer location (see at least col. 136, lines 7-10 "**STOREDEC, is a pre-decrementing store that pushes the IP value onto the stack segment (SS) at the offset specified by the stack pointer ESP**");
- loading a code segment (CS) value stored one location after the stack pointer location into the stack pointer location (see at least col. 135, lines 48-49 "**the current offset into the code segment (EIP) is pushed onto the stack**");
- loading an EFLAGS values stored two locations after the stack pointer location into one location after the stack pointer (see at least col. 91, lines 13-16 "**whether addressing is physical or virtual, the floating point stack pointer, and the full/empty state of floating-point register, and operand sizes are encoded in segment descriptors, the EFLAGS register**");

- loading a base pointer into two location after the stack pointer location (see at least col. 132, lines 7-9 "**the eight STOREDEC instructions 951 push the eight general purpose registers EAX, ECX, EDX, EBX, ESP, EBP, ESI and ECI onto the stack**"); and
- loading the base pointer into a base pointer register (see at least col. 34, lines 9-10 "**EBP register of the X86 architecture are mapped by converter hardware 136 to register R53**").

As per claim 28:

Yates discloses:

- decrementing the stack pointer location by one location (see at least col. 132, lines 43-44 "**the register carries the intermediate decrementing of the stack pointer**"); and
- issuing a return from interrupt instruction ("**a trap instruction transfers control to an exception handler**" Col 94, line 15-16).

As per claim 29:

Yates discloses:

- wherein the instruction pointer is loaded at the stack pointer location (see at least col. 136, lines 7-10 "**STOREDEC, is a pre-decrementing store that pushes the IP value onto the stack segment (SS) at the offset specified by the stack pointer ESP**").

As per claim 30:

Yates discloses:

- wherein emulating the instruction comprises emulating a pop1 instruction (see at least col. 58, line 1 "**when an X86 POPF instruction is emulated**").

As per claim 31:

Yates discloses:

- obtaining a stack pointer location, wherein the stack pointer location corresponds to a location in the stack frame (see at least col. 132, lines 7-9 "**the eight STOREDEC instructions 951 push the eight general purpose registers EAX, ECX, EDX, EBX, ESP, EBP, ESI and ECI onto the stack**");
- loading the base pointer obtained from three location after the stack pointer location into a base pointer register (see at least col. 34, lines 9-10 "**EBP register of the X86 architecture are mapped by converter hardware 136 to register R53**");
- loading an EFLAGS values stored two locations after the stack pointer location into one location after the stack pointer (see at least col. 91, lines 13-16 "**whether addressing is physical or virtual, the floating point stack pointer, and the full/empty state of floating-point register, and operand sizes are encoded in segment descriptors, the EFLAGS register**");

- loading a code segment (CS) value stored one location after the stack pointer location into the stack pointer location (see at least col. 135, lines 48-49 "**the current offset into the code segment (EIP) is pushed onto the stack**");
- incrementing an instruction pointer to obtain an incremented instruction pointer (see at least col. 122, lines 38-40 "**the instruction pointer (IP) value is incremented by the length count after each instruction that is marked with an end-of-recipe marker**"); and
- loading the incremented instruction pointer in the stack frame at one location before the stack pointer location (see at least col. 136, lines 7-10 "**STOREDEC, is a pre-decrementing store that pushes the IP value onto the stack segment (SS) at the offset specified by the stack pointer ESP**").

As per claim 32:

Yates discloses:

- incrementing the stack pointer location by one location (see at least col. 144, lines 65-67 "**when an X86 program changes its stack, both the stack segment descriptor (SS) and the offset may need to be changed**"; also see at least Col 145, line 2 "...**the instruction that loads the stack offset into the stack pointer**"); and
- issuing a return from interrupt instruction (see at least col. 94, lines 15-16 "**a trap instruction transfers control to an exception handler**").

As per claim 33:

Yates discloses:

- wherein the instruction pointer is loaded at the stack pointer location(see at least col. 136, lines 7-10 "**STOREDEC, is a pre-decrementing store that pushes the IP value onto the stack segment (SS) at the offset specified by the stack pointer ESP**").

As per claim 34:

Yates discloses:

- wherein emulating the instruction comprises emulating a leave instruction (see at least col. 145, lines 7-10 "**at the end of emulating the move or pop instruction**").

As per claim 35:

Yates discloses:

- obtaining a stack pointer location, wherein the stack pointer location corresponds to a location in the stack frame (see at least col. 132, lines 7-9 "**the eight STOREDEC instructions 951 push the eight general purpose registers EAX, ECX, EDX, EBX, ESP, EBP, ESI and ECI onto the stack**");
- loading the base pointer obtained at the base pointer location into base pointer register (see at least col. 34, lines 9-10 "**EBP register of the X86 architecture are mapped by converter hardware 136 to register R53**");

- loading an EFLAGS values stored two locations after the stack pointer location into one location after the stack pointer (see at least col. 91, lines 13-16 "**whether addressing is physical or virtual, the floating point stack pointer, and the full/empty state of floating-point register, and operand sizes are encoded in segment descriptors, the EFLAGS register**");
- loading a code segment (CS) value stored one location after the stack pointer location into the stack pointer location (see at least col. 135, lines 48-49 "**the current offset into the code segment (EIP) is pushed onto the stack**");
- incrementing an instruction pointer to obtain an incremented instruction pointer (see at least col. 122, lines 38-40 "**the instruction pointer (IP) value is incremented by the length count after each instruction that is marked with an end-of-recipe marker**");
- loading the incremented instruction pointer in the stack frame at one location before the stack pointer location (see at least col. 136, lines 7-10 "**STOREDEC, is a pre-decrementing store that pushes the IP value onto the stack segment (SS) at the offset specified by the stack pointer ESP**"); and
- loading the instruction pointer at three location before the base pointer (see at least col. 135, line 49 "**EIP is pushed onto the stack**").

As per claim 36:

Yates discloses:

- setting the stack pointer location to three locations before the base pointer location (see at least col. 132, lines 7-9 "**the eight STOREDEC instructions 951 push the eight general purpose registers EAX, ECX, EDX, EBX, ESP, EBP, ESI and ECI onto the stack**" – **the idea is to obtain the stack pointer**);
- incrementing the stack pointer location by one location (see at least col. 144, lines 65-67 "**when an X86 program changes its stack, both the stack segment descriptor (SS) and the offset may need to be changed**"; also see at least col. 145, line 2 "...**the instruction that loads the stack offset into the stack pointer**"); and
- issuing a return from interrupt instruction (see at least col. 94, lines 15-16 "**a trap instruction transfers control to an exception handler**").

Conclusion

3. **THIS ACTION IS MADE FINAL.** Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any

extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the mailing date of this final action.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Phillip H. Nguyen whose telephone number is (571) 270-1070. The examiner can normally be reached on Monday - Thursday 10:00 AM - 3:00 PM EST.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Wei Y. Zhen can be reached on (571) 272-3708. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

PN
6/29/2007

WEI ZHEN
SUPERVISORY PATENT EXAMINER